AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Bachelor Project
Theory documentation

by

Daniel Christopher Biørrith, 201909298

Lukas Koch Vindbjerg, 201906015

Bachelor's Thesis

in

Computer Engineering

Supervisor: Kim Bjerge

Aarhus University
Department of Electrical and Computer Engineering

June 15, 2022

Character count: 18028 (of total 72.000)

Contents

Chapter 1

Acquisition - Physical image theory

A digital microscope works by applying the optics of a traditional microscope and the sensor of a digital camera. An image is obtained in much the same way, as a conventional digital camera, which means similar parameters can be considered for the quality of the final image. Primarily, this can be facilitated as three main topics; the focus of the lens, the time the sensor is exposed to light and the sensor's sensitivity to light.

The first of these parameters to consider is how to keep the image sharp which is greatly affected by the optics of the lens. More specifically, it is controlling if an object is in focus. As the light from the object passes into the lens it is refracted by the lens as shown in figure 1.1.

Here S_1 is the distance between the object and the camera lens, S_2 is the distance between the lens and the image and F is the focal length. Modifying any of these parameters (or the lens shape) will put the image either in focus or out of focus.

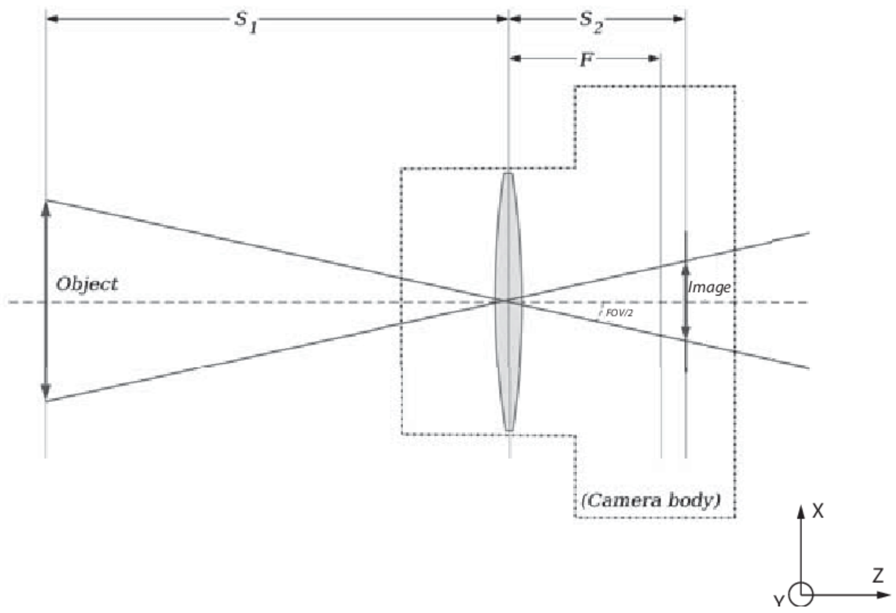


Figure 1.1: Schematic overview of a camera with lens[1, Figure 2]

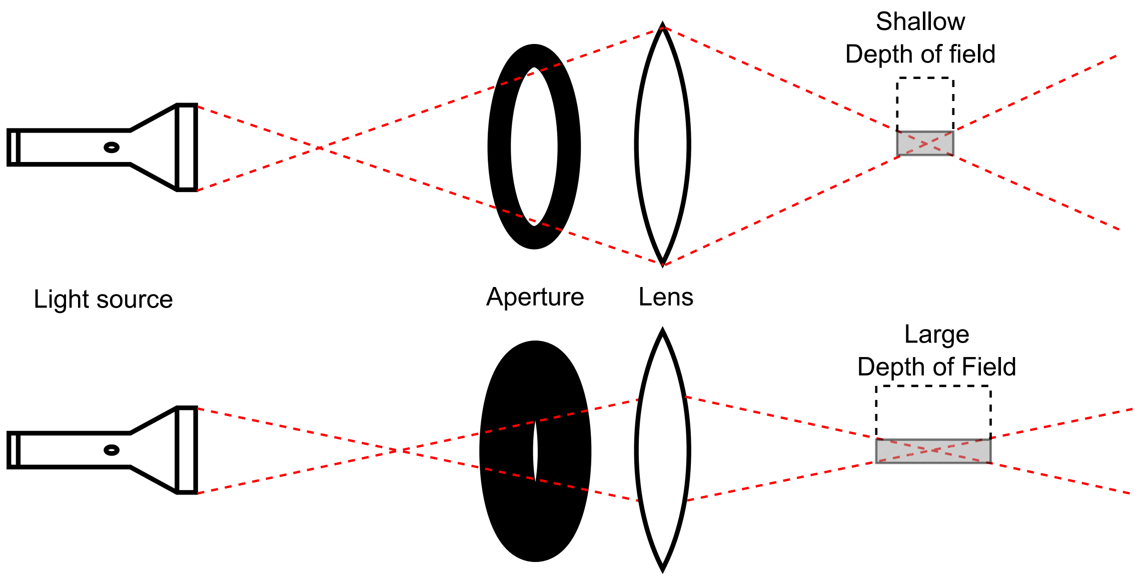


Figure 1.2: Aperature diagram

The object of interest appears the sharpest once the distance S_1,S_2 and the focal length fit together such that the same spot from the real world maps to the same spot on the sensor.

However, there is an additional parameter that contributes to the sharpness of an image; the aperture. The aperture is the parameter that describes the physical size of the opening of the lens. The aperture controls how the depth of field changes which then determines how far in front and behind the object in focus is also in focus[2]. This is illustrated in figure 1.2 where the depth of field becomes shallower the larger the aperture is. Additionally, these parameters become more and more sensitive the greater the magnification of the image. In turn, this greatly amplifies the effect of any small changes to the parameters. For the implementation of the DFI system, this has the impact that these settings have to be calibrated with quite a large precision. Nevertheless, it is not unlikely that some granular material is too large for the system. Meaning that even with the smallest aperture, the depth of field of the microscope is too shallow resulting in either the top or bottom of the material inevitably getting out of focus.
The final aspect to take into account for the DFI system is the interaction with the camera sensor.

Regarding the sensor there are two primary adjustable parameters; the time the sensor is exposed to light and the sensor's sensitivity to light. The time the sensor is exposed to light is referred to as the exposure time and determines for how long the photon detecting pixels will be reacting to incoming photons. Increasing the exposure time will then in turn collect more information from the image. However, for capturing many images, increasing the exposure time has the downside of being slower as the camera has to pause for longer for each image. Additionally, the image becomes more sensitive to instabilities and vibrations as any movement of the sensor, while exposed, can make points in an image map to different pixels, essentially blurring the image.

Fortunately, these downsides can be counteracted by the final adjustable parameter, the sensor's sensitivity to light. This parameter simply works as an amplifier of the digital value for each sensor pixel. This results in the possibility of getting an image of similar brightness as one with a longer exposure time while mitigating the previously mentioned drawbacks. The issue of increasing the sensitivity, however, is that it will likewise amplify any noise in the image. Having noise in an image is inevitable which is why having a lower sensitivity is usually desirable. It is evident that any of these parameters require adjustment of the others for the desired image quality. Balancing the exposure time and the sensitivity of the sensor becomes a crucial design decision which has to be made when developing a camera-based system.

It is worth noting that all of the above-mentioned aspects of a digital microscope are way more complex than explored in this chapter. For instance, a single microscope lens can consist of dozens of focusing elements and the sensitivity of the sensor is way more complicated than just being a value multiplier. For a system like the DFI system, the parameters and understanding described at the level above are sufficient for understanding the design decisions and implementations.

Chapter 2

Digital Image Processing

2.1 Digital image representation

Typically, a digital image can be understood as a 2-dimensional grid (x, y) where each position represents a discrete pixel. Looking at a grey-scale image this pixel has a value within a certain range. Essentially a value of 0 is a black pixel and a value of either 1 or 255 (depending on the format) is a white pixel. For a coloured picture, a common representation is the RGB colour model. This model has much the same structure as a grey-scale image, except the value of a pixel is represented as an array with a length of three. This array refers to the three colour channels; red, green and blue. For each of the channels, the value now represents how much of the respective colour is shown instead of it just being light or dark. A standard notation of an image could thus be:

$$(x, y, [R, G, B])$$

This is one of the more common representations of a digital image. However, there are many other ways to represent a digital image. Each representation has its one advantage and being able to transition between them is the basis of a lot of image processing.

For example, a very useful representation of an image is a binary image. A binary image is shown in figure 2.1c which illustrates that every pixel is either completely black or completely white. The way a binary image is created is first by converting it from a coloured image to a grey-scale image. This is simply done by representing each pixel as the weighted sum of each colour channel for that respective pixel. Converting the image to a binary image is then done by checking if each pixel is above a certain threshold. For figure 2.1c for instance, each pixel of figure 2.1b with a floating point value less than 0.45 has been turned black and each pixel with a greater value has been turned white.
Binary images are useful for many operations such as morphological operations or segmentation which will be explored later. There are many ways a binary image can be generated which can highlight various aspects. A useful method is the Otsu method which finds a fitting threshold for a given image[3, p. 752].

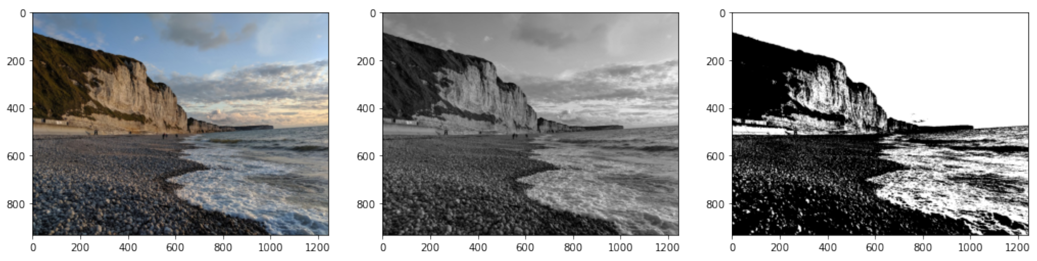

(a) Coloured image (b) Grey-scale image (c) Binary image

Figure 2.1: Colour representation of image

2.2 Morphological processes

Morphological operations are useful methods to manipulate an image concerning regions and noise. When referring to morphology in digital image processing, it applies mathematical morphology which is a branch of set theory[3, p. 635]. This makes morphology a versatile and powerful tool when processing digital images. In general, morphology is concerned with the representation and descriptions of shapes and regions of the image. While morphological operations can be applied to not only binary images, but to grey-scale images and coloured images as well. However, morphological operations done on binary images require a lot less complexity without compromising much for applicability. In binary images, the sets of objects are represented in a 2D integer space, $\mathbb{Z}^2$. The element (represented as a tuple) has the coordinates of a foreground pixel in an image.

There are two fundamental morphological operations; erosion and dilation. The operations are expressed in terms of a set A of foreground elements (or simply put, a binary image) and a structured element B where $A, B \in \mathbb{Z}^2$. The general notation of an erosion operation between A and B is written as: $A \ominus B$ and is defined as

$$A \ominus B = \{z | B_z \subseteq A\} \tag{2.1}$$

where z is the foreground values[3, p. 639]. Equation 2.1 basically expresses that the erosion of A by B is all the points z where B translated by z is contained in A. An illustration of this is shown in figure 2.2. Here we can see that the erosion operation between the original image by the structured element results in a thinner line at the top and then a smaller box below. This is due to what is meant by B_z which is B translated by z. The operation takes the structured element and maps it to all positions of z which is all the foreground values. This simply means that the center of the structured element - represented by the black dot - is placed on one of the foreground values of the original image. If then the entire structured element is contained in foreground values ($B_z \subseteq A$) then the center of the resulting image will likewise be a foreground element. Consequently, if some part of the structuring element is not a foreground element then the center of the resulting image will not be a foreground element. This is evident in figure 2.2 as the structured element extends one pixel out from the center in all directions. This means that the erosion operation with this structured element essentially 'trims' the edges as translating the structured element onto one of the edge foreground values will not give a resulting foreground value. The shape of the structured element can be any desired shape or

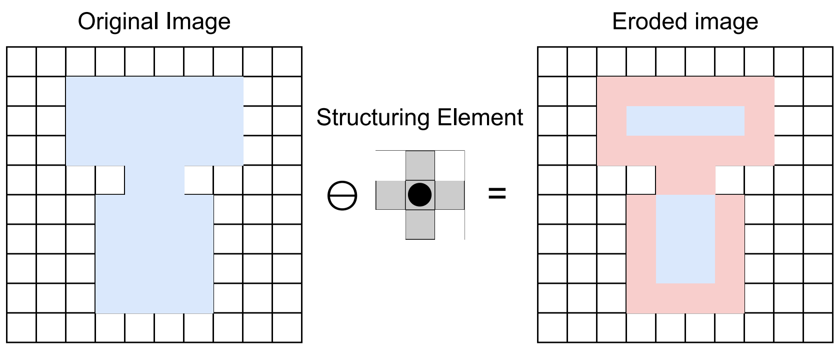


Figure 2.2: Example of erosion operation on an image. The blue pixels of the left most image represent foreground pixels. The grey pixels represent the structured element in the center image. Finally, in the rightmost image, the blue pixels represent the remaining foreground elements and the red pixels represent the removed foreground elements after the operation.

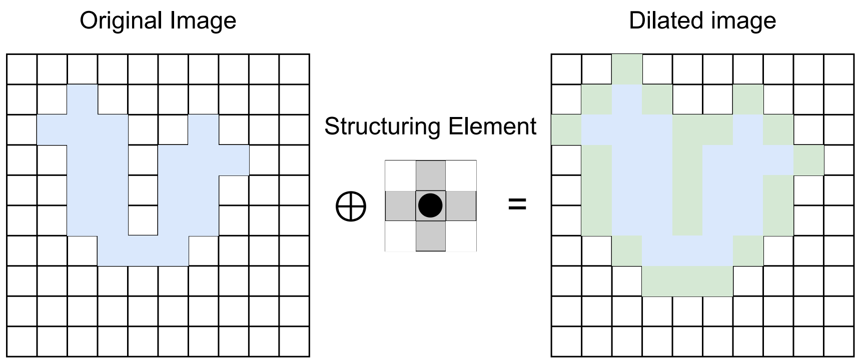


Figure 2.3: Example of dilation operation on an image. Similar to figure 2.1 the blue pixels represent foreground elements. The grey pixels of the middle image represent the structuring element. Finally, the green pixels represent the new foreground pixels after the operation.

size which is one of the reasons morphological operations are so versatile. Depending on the desired use, the structured element can e.g. be quite spherical to remove lines or scratches. Likewise, a more narrow and long structured element could be used to isolate lines or a grid pattern.

The second morphological operation is dilation. The general concept is quite similar where the major difference is when and how the structured element is kept as a foreground element. Again, with an image A and a structured element B where $A, B \in \mathbb{Z}^2$ the notation for dilation is $A \oplus B$ and is defined as

$$A \oplus B = \{z | [(\hat{B})_z \cap A] \subseteq A\} \tag{2.2}$$

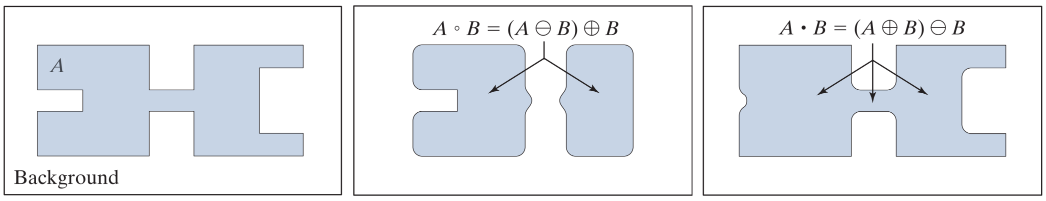
Equation 2.2 The difference between erosion and dilation is that if the center of the structured element is mapped to a foreground element, then the entire structured element will appear as a foreground element on the resulting image. In figure 2.3 it is shown how the dilation operation changes an image. It works much as an opposite to erosion but they are not exactly the inverse of one another which is why they are often used together in operations called opening and closing.
Opening and closing are morphological operations where erosion and dilation are

used consecutively. Opening is denoted as $A \circ B$ and closing is denoted as $A \bullet B$ and they are defined as:

$$\text{Opening:} \quad A \circ B = (A \ominus B) \oplus B \tag{2.3}$$

$$\text{Closing:} \quad A \bullet B = (A \oplus B) \ominus B \tag{2.4}$$

The effects of these operations are quite apparent from their names. Opening will take a shape and open up any thin areas and closing will fill out any gaps. This can be seen in figure 2.4 where the same picture turns out completely different depenging on the operation. These operations can thus be very useful if the goal is e.g. isolate


A
Background
$A \circ B = (A \ominus B) \oplus B$
$A \bullet B = (A \oplus B) \ominus B$

(a) Original image (b) Opened image (c) Closed image

Figure 2.4: Opening and closing operations applied on the same figure

two spheres in the same picture where opening will separate them.

2.3 Segmentation

Segmentation is the process of identifying unique elements of an image and isolating them. There are many ways to segment an image depending on both the input and the intended use. One of the more simple methods is the threshold based segmentation[3, p. 742]. This method works by subdividing images directly into objects based on the characteristic values of the pixels. For this reason, binary images are a big part of threshold based segmentation. Here the foreground pixels will be grouped together with neighbouring foreground pixels to generate a set that contains all the pixels of an isolated segment.

2.4 Stitching

Image stitching is the process of combining multiple overlapping images into a single image. The general process is to first gather registration data and then composite this data into a larger panoramic image[4]. The registration data is generated with feature extraction by first detecting features and then matching features. Subsets of these features are then used to build a panoramic section. With the registration data generated the compositing process can be executed. Here warping, exposure difference and scaling are evaluated and compensated for such that a final seemliness image can be returned.

Chapter 3

Classification

This section will describe what a convolutional neural network (ConvNet) is, how it works and why it is state-of-the-art in regards to classifying images. It will first present and explain a standard neural network as it stands as a basis for a ConvNet. Afterwards, it will describe ConvNets and finally explain the specific type of ConvNet used for creating a model.

3.1 Neural network - multilayer perceptron

In order to understand ConvNet, one must first understand what a normal neural network (NN) is [5]. A neural network is an adaptive form of learning with basis functions. It is composed of layers with a connection between them, which holds a certain weight. It receives x number of inputs and can give y number of outputs, where $x, y \in \mathbb{R}$. An example of a NN is depicted in figure 3.1.

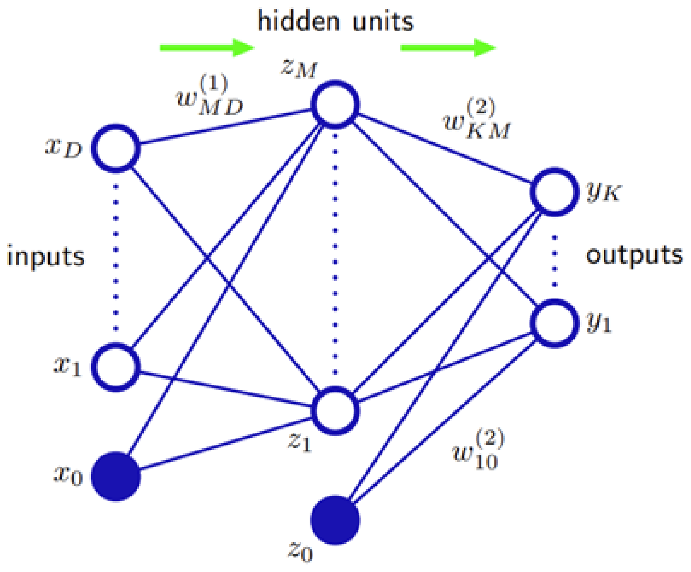


Figure 3.1: An example of a two layered neural network [5, Fig 5.1]

3.1.1 Architecture

A NN is composed of neurons, also known as hidden units, which are located in each layer. These defined the width of a layer, as the number of neurons is exactly that. Every neuron in a given layer is connected to the every neuron in the previous layer (or the input) and every neuron, in the next layer (or the output) with a *weight*, which is nothing but a constant value. With this, the input value to each neuron can be defined as:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(L-1)} x_i + w_0^{(L-1)}$$

Here, j is defined as $j = 1, ..., M$, where M is the number of neurons in a layer. D is the dimension of the previous layer (as it connects to each neuron in the previous layer), $w_{ji}^{(L)}$ is the weight of all the connections, and (L-1) represents the previous layer. Finally, $w_0^{(L-1)}$ is a certain bias each node can have. Next we define the neurons output. In order to do that, we first have to introduce activation functions.

3.1.2 Activation functions

In order to avoid linearity, the activation function has to be a non-linear basis function. The reason for this can be found in [5, page 229]. Historically, two actiavtion functins have been used. These are the sigmoid function and the ReLU function. They are defined as follows:

$$ReLU(a) = max(0, a)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Nowadays, by far the most used of the two is the ReLU function, for reasons stated here[6]. The activation function helps make it only the relevant neurons that are activated, making the network faster and more accurate.

3.1.3 Neuron output

With an activation function defined, we are ready do define the output of a neuron. This is defined as:

$$z_j = h(a_j)$$

Where $h(\cdot)$ is a non-linear activation function, typically ReLU as previously stated. This output now goes to all neurons in the next layer.

A neural network is feed forward, meaning no neurons can ever provide a value back to the previous layer. Feed forward is necessary to ensure that the outputs are deterministic functions of the hidden input, meaning no closed direct cycles appear. When reaching the output layer, there are multiple choices for an output activation function. As we develop an binary output (two classes, foraminifera and sand), the only one relevant for us is the Sigmoid class as defined earlier.

3.1.4 Error Backpropagation

Error backpropagation, or simply known as backprop, is when you use local message passing scheme in which information is sent alternately forwards and backwards through the network. A NN is 'trained' on a dataset through a technique called backwards propagation. Here, you feed the network input values, by which you get certain output values, as weights are set to an initial value. You compare the provided output values with the actual wanted values, and optimize the weight to get the wanted output by error backpropagation. Without going into details of the method, it makes use of the gradient vector to go towards the minimum error. This is called **gradient descent**.

3.1.5 Invariance in data

A big problem with neural networks is invariance in data. You wish for data to be classified the same regardless of position and size of the object. Changes in the those two are called translation invariant and scale invariant, respectively. Such transformations produce significant change in the input data. Though it's a change in input, it should produce the same result. It is possible to train a neural network with sufficiently large training set to circumvent this (at least approximately), however it's very costly. Therefore, there are ways to work around this.
Taken from the previously cited book, these are the two most relevant methods to circumvent this:

1. "The training set is augmented using replicas of the training patterns, transformed according to the desired invariances. For instance, in our digit recognition example, we could make multiple copies of each example in which the digit is shifted to a different position in each image.". This is also known as augmenting the data, and will be implemented in the created model.

2. "The final option is to build the invariance properties into the structure of a neural network (or into the definition of a kernel function in the case of techniques such as the relevance vector machine). One way to achieve this is through the use of local receptive fields and shared weights, as discussed in the context of convolutional neural networks.". This is e

Both of these options are optimal for our case and will be implemented when training our model. Other methods to further improve the network, which will be implemented but not explained here, are dropout and Early stopping.

3.2 Convolutional neural network

Next we talk about convolution neural networks (ConvNet). ConvNets are a form of neural networks, where additional convolutional and pooling layers have been introduced. These layers help define certain features of images, and are immensely helpfull when experiencing data invariance. The convolution layer helps extract the features by performing the mathematical convolution operation on the input [7].

The pooling layer helps reduce the dimensions of an input by collection data from smaller regions. There exists two types of pooling, average and max pooling. The two are depicted in figure 3.2.

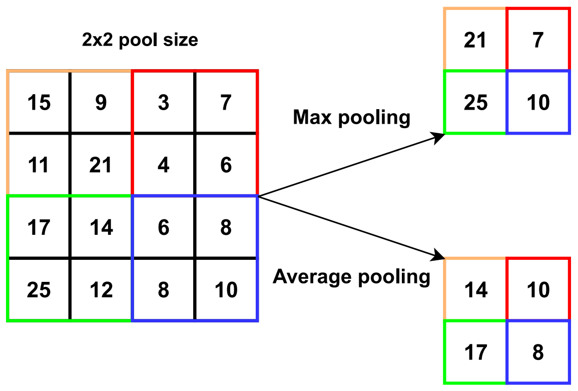


Figure 3.2: Max and average pooling with a stride of 2

The figure demonstrates how 2x2 size boxes reduces the dimensions of the left box, while keeping the key values into the smaller dimensions. Generally, max pooling help reduce the noise in the image, while average pooling keeps the information of all inputs.

3.3 EfficientNet

EfficientNet[8][9] is a specific type of ConvNet, first introduced in 2019. With it, a new way method to scale up ConvNets was proposed.

Scaling of models in terms of either width (more Neurons in each layer), depth (more layers), or higher resolution (larger inputs) are ways to improve the performance of a ConvNet. The conventional practice is to arbitrary scale these factors, most commonly by scaling depth by adding more layers, or by width by adding more neurons to certain layers. However, this is typically done by arbitrary manual tuning without any structure to it.

The creators sought to find a new way to scale ConvNets. In their search they found the key observation listed below:

- "Scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models". It's demonstrated in their paper how only increasing one dimension will improve accuracy, but quickly stagnate in performance improvement.

- "In order to pursue better accuracy and efficiency, it is critical to balance all dimensions of network width, depth, and resolution during ConvNet scaling.". With this observations, they saw that a systematic scaling would improve performance more than only scaling one parameter.

With this, they realized it's more optimal to uniformly scale depth, width and resolution, rather than only a single of those. With this, they proposed a method for balancing the scale of all parameters, simply by scaling with a constant ratio. The proposed method was called *compound scaling method*. The visual difference between it and other scaling methods is depicted in figure 3.3.

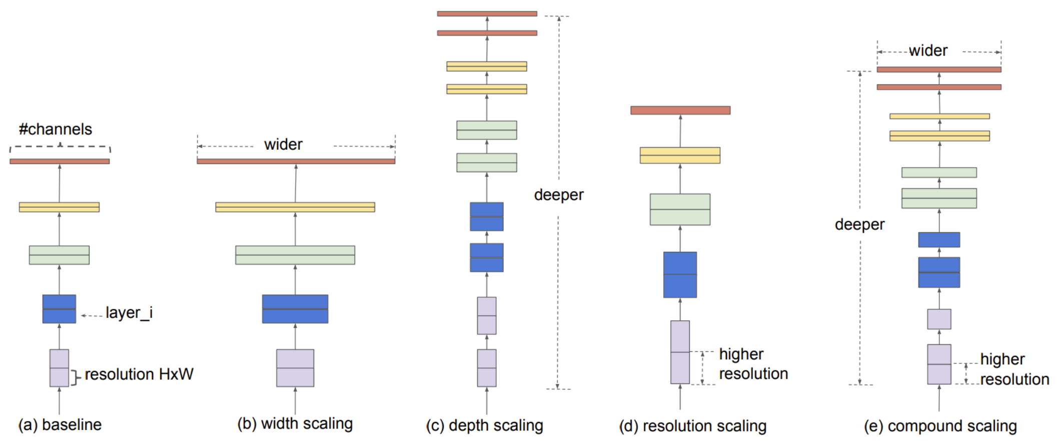


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Figure 3.3: Model scaling as proposed by the EfficientNet developers. [8, Figure 2]

Compound scaling introduces three constant values, α, γ, and β, representing the depth, width, and resolution respectively. Additionally, they introduced ϕ, which is the compound coefficient. The compound coefficient is user-specified, representing how much the model is to be scaled up based on the available computational resources. α, γ, and β specify how to assign those resources. Now, if you wish to use 2^N more computational resources, you scale the size to α^N γ^N and β^N. Importantly, the size of those values are to satisfy the following ecuation:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

If they do, any new ϕ will ensure the number of FLOPS will increase by 2^ϕ, for reason found in the paper. In order to test this method of scaling, EfficientNetB0 was created. The architecture of the network is shown in figure 3.4. It served as a baseline network and was to be tested upon with the compound scaling method. From it, EfficientNetB1 through EfficientNetB7 were created. The one used in this project will be the B7 version, as it's the most precise, though it's also the largest.

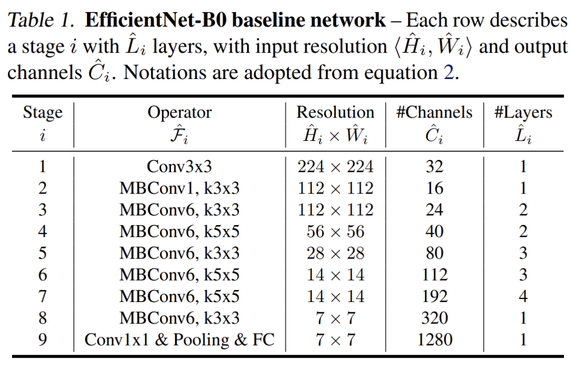


Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 3.4: Architecture of EfficientNet-B0 baseline network. [8, Table 2]

Bibliography

[1] B. Ribbens, "Development and validation of a time domain fringe pattern analysis technique for the measurement of object shape and deformation," Ph.D. dissertation, Mar. 2015. [Online]. Available: https://www.researchgate.net/publication/273652708_Development_and_validation_of_a_time_domain_fringe_pattern_analysis_technique_for_the_measurement_of_object_shape_and_deformation.

[2] K. R. Spring and M. W. Davidson. "Depth of field and depth of focus." (2022), [Online]. Available: https://www.microscopyu.com/microscopy-basics/depth-of-field-and-depth-of-focus (visited on 06/03/2022).

[3] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4. edition. Pearson Education Limited, 2018, ISBN: 9780133356779. [Online]. Available: https://www.pearson.com/us/higher-education/program/Gonzalez-Digital-Image-Processing-4th-Edition/PGM241219.html?tab=resources (visited on 03/29/2022).

[4] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International Journal of Computer Vision*, vol. 74, no. 59-73, 2007. DOI: /10.1007/s11263-006-0002-3. [Online]. Available: https://rdcu.be/cPzwI (visited on 06/13/2022).

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*, 2. edition. Springer New York, 2006, ISBN: : 978-1-4939-3843-8. [Online]. Available: https://link.springer.com/book/9780387310732 (visited on 06/10/2022).

[6] A. Thakur, "Relu vs. sigmoid function in deep neural networks," 2022. [Online]. Available: https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks-Why-ReLU-is-so-Prevalent--VmlldzoyMDk0MzI (visited on 06/12/2022).

[7] Wikipedia. "Convolution." (), [Online]. Available: https://en.wikipedia.org/wiki/Convolution (visited on 06/14/2022).

[8] M. Tan, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1905.11946v5 (visited on 06/13/2022).

[9] Papers with Code, "Efficientnet," [Online]. Available: https://paperswithcode.com/method/efficientnet (visited on 06/13/2022).

[10] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, *et al.*, "Scikit-image: Image processing in python," *PeerJ*, vol. 453, no. 2, 2014. DOI: /10.7717/peerj.453. [Online]. Available: https://peerj.com/articles/453/.

[11] N. Lang. "Using convolutional neural network for image classification." (2021), [Online]. Available: https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4 (visited on 06/12/2022).